**Intelligent CS 8B?**

An *object* is structured data that is alive, responsible, and *intelligent*.

Sound too friendly?

This week’s objects and classes will be just the opposite ...

---

This lecture and assignment are **OPTIONAL**!

---

Thanks to Zach Dodds at Harvey Mudd for the slides
Deep Blue was a chess-playing computer developed by IBM. On May 11, 1997, the machine, with human intervention between games, won the second six-game match against world champion Garry Kasparov by two wins to one with three draws.[1] Kasparov accused IBM of cheating and demanded a rematch, but IBM refused and dismantled Deep Blue.[2] Kasparov had beaten a previous version of Deep Blue in 1996.
Strategic thinking!

Java *had* no Connect-four datatype…

I feel ahead of the game here…

| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | |X| | | |
| |X| |X|O| | |
|X|O|O|O|X| |O|

How many moves ahead might we have to look?

I feel ahead of the game here…

May the best alien win

| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | |X| | | |
| |X| |X|O| | |
|X|O|O|O|X| |O|

… but you've already corrected that!

DEMO!

Now, *may the best machine win*! 
The `ConnectFourPlayer` class

What data does a computer AI player need?

What data does a computer AI player need?

DATA MEMBERS

- `char checker`  
  checker, O or X

- `String tieBreakType`  
  tiebreakType

- `int ply`  
  moves to look ahead

surprisingly little!

How about knowledge about its opponent?
The **C4Player** class

**C4 Board**

Board(int width, int height)

boolean allowsMove( int col )

void addMove( int col, char checker )

**void delMove( int col )**

String toString()

boolean isFull( )

boolean winsFor( char checker )

void hostGame( )

**C4Player**

C4Player( char ch, String tbt, int plyIn )

String toString()

cchar oppCh()

double scoreBoard(C4Board b)

double[] scoresFor(C4Board b)

int tiebreakMove(double[] scores)

int nextMove(C4Board b)
Make no mistake about it: computers process numbers - not symbols.

Computers can only help us to the extent that we can arithmetize an activity.

- paraphrasing Alan Perlis
Assigns a \textit{score} to any board, \( b \)

A simple system:

\begin{tabular}{ccc}
\textbf{Score for} & \textbf{100.0} & \textbf{50.0} & \textbf{0.0} \\
\textbf{for a win} & \textbf{for anything else} & \textbf{for a loss} \\
\end{tabular}

\begin{enumerate}
\item \( \text{Score for } \bullet \rightarrow 100.0 \) \hfill \text{Score for } \bullet \rightarrow 50.0 \\
\item \( \text{Score for } \circ \rightarrow 0.0 \) \hfill \text{Score for } \circ \rightarrow 50.0 \\
\end{enumerate}
Assigns a score to any board, \( b \)

A simple system:

<table>
<thead>
<tr>
<th>Score</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.0</td>
<td>for a win</td>
</tr>
<tr>
<td>50.0</td>
<td>for anything else</td>
</tr>
<tr>
<td>0.0</td>
<td>for a loss</td>
</tr>
</tbody>
</table>

Implementing...

Why do we need to input the board? We never had to do that before!

How can there be no 'X' or 'O' input?

What methods that already exist will come in handy?

This isn't looking very far ahead!
Scoring *moves* at 0 ply

"Ply" is the number of moves to look ahead...

What would the 0-ply scores be for each *column* (move)?

0 ply is a Zen-like approach: *exist only in the present*

We use \(-1\) as the score into a full column.

```javascript
o0.scoreBoard( b42 )
o0.scoresFor( b42 )
```
Scoring *moves* at 1 ply

What would the 1-ply scores be for each *column* (move)?

A 1-ply lookahead player will "see" an impending victory.

"Gotcha!"

1-ply scores for 0

1-ply means 1 move is made!

to move
Seeing the recursive structure

1-PLY SCORES FOR ⃝

-1 SD

1-PLY MEANS 1 MOVE IS MADE!

0-PLY SCORES FOR ⚫

0-PLY MEANS 0 MOVES ARE MADE!
Seeing the recursive structure

1-ply scores for ●

-1  50  50  50

1-ply means 1 move is made!

0-ply scores for ●

-1  0  0  0  0  0  0  0

0-ply means 0 moves are made!

to move

to move
Scoring moves at 2 ply

What would the 2-ply scores be for each column (move)?

A 2-ply lookahead player will see a way to win or block the opponent's win

"Gotcha!" + "Uh Oh..."

2-ply scores for:

-1 50 50 50 100 50 50

2-ply means 2 moves are made!

What about 3-ply?
Scoring *moves* at 2 ply for •

What would the 2-ply scores be for each *column* (move)?

A 2-ply lookahead player will see a way to win or block the opponent's win

"Gotcha!" + "Uh Oh..."

What about 3-ply?
**scoreBoard** vs. **scoresFor**

**double**

**scoreBoard**(C4Board b) looks ahead 0 moves

0-ply only

it **scores** the **board** as it stands **now**

**double CJ**

**scoresFor**(C4Board b) returns a LIST of scores: *one for each column a player might move next…*

0-ply

it gives a separate **score for** each column and considers future moves up to **n-ply**

1-ply

2-ply

...
Fill in the score for each column at each ply up to 3.

The same move is evaluated at each ply... it's just evaluated farther into the future!

Each row is different in at least 1 score…

0-ply scores for 'O':
Looks 0 moves into the future

1-ply scores for 'O':
Looks 1 move into the future

2-ply scores for 'O':
Looks 2 moves into the future

3-ply scores for 'O':
Looks 3 moves into the future
(0) Suppose you're playing at 2 ply...

(1) Make ALL moves!

scoresFor's idea

(this) 'X'
(0) Suppose you're playing at 2 ply...

(1) Make ALL moves!

scoresFor's idea

(0) Suppose you're playing at 2 ply...
(1) Make ALL moves!
(0) Suppose you're playing at 2 ply...
(1) Make ALL moves!
(2) Ask OPPONENT its scoresFor at ply-1
(3) Compute which score the opp. will take
(4) Compute what score you get...

these are all of the opponent's calls to scoresFor!
(0) Suppose you're playing at 2 ply...

(1) Make ALL moves!

(2) Ask OPPONENT its scoresFor at ply-1

(3) Compute which score the opp. will take

(4) Compute what score you get...

these are all of the opponent's calls to scoresFor!